Counterfactual Fairness in deep generative image models

Sairah Joseph, Shubanshu Gupta, Neel Bhave, Virender Singh, Oj Sindher

Problem Statement

In this tutorial we implement a Deep Structural Causal Model for Tractable Counterfactual Inference (Pawloski, Castro, et. al.) on the Google Cartoon Dataset of artist-created randomly generated faces. Future work will show how this technique is useful for evaluating counterfactual fairness in deep generative models of faces.

The Google Cartoon Dataset

The Google Cartoon Dataset is a dataset of artist-created components combined randomly to generate thousands of cartoon faces. From the Google Cartoon Dataset: Each image of a cartoon face is composed of **16 components** that vary in **10 artwork attributes**, **4 color attributes**, and **4 proportion attributes**. Each of these components and their result in approximately **250** cartoon component artworks and **~10^{13}** possible combinations. Any random selection of attributes produces a visually appealing cartoon without any misaligned artwork; this sometimes involves **handling interaction between attributes**. For example, the proper way to display a “short beard” changes for different face shapes, which requires the artist to create a “short beard” artwork for each face shape.
The attributes were randomly combined to produce n=10k and n=100k datasets.

Variational AutoEncoders

The variational Autoencoder is the fundamental block for our deep generative causal model.
Variational Autoencoder is a type of autoencoder in which the latent representation is not reduced to a single point but a distribution. This distribution over latent space helps us to generate some new data points. The relation between the variational inference and the regularization method makes it variational.

In general, the encoding part in the variational autoencoder gives a latent representation for the mean and the standard deviation and then using that for the normal distribution, we sample the latent space datapoints. We then take the random point from the latent space and then decode it to get a new datapoint in the real image space.

We use Bayes’ Theorem to arrive at a probability distribution for the latent variables $\bold z$ given the variables of interest $\bold x$:

$$P_\theta(z|x) = \frac{P_\theta(x|z)P_\theta(z)}{P_\theta(x)}$$

We use the variational distribution $Q(z | x)$ as an estimator of the true probability distribution $P(z | x)$ and learn it using the Kullback-Leibler divergence as defined below:

$$D_{KL}\left(Q_\phi(z|x)\|P_\theta(z|x)\right) = \mathbb{E}_{z\sim Q}\left[\log Q_\phi(z|x) - \log P_\theta(x|z) - \log P_\theta(z)\right] + \log P_\theta(x)$$

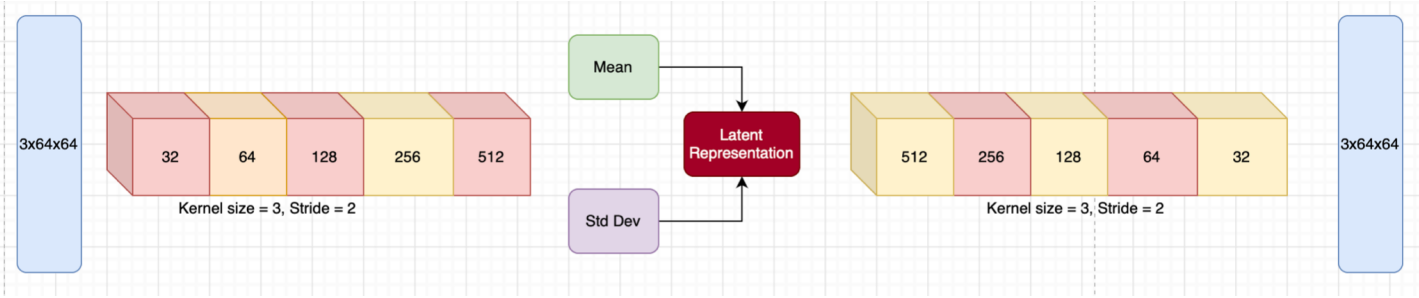


Figure 1 : The architectural diagram for the variational autoencoder

For our project, the variational autoencoder works as the sample test standard that we want to achieve using the Normalizing flows. The ideal scenario would be to have the reconstructed images from the normalizing flows to be of the comparable quality as that of the variational autoencoder.
The above diagram shows the simple variational autoencoder we are using, The image size taken as the input is of size 3x64x64 and the encoding layers have the convolutional layers of size [32,64,128,256,512] respectively. Similarly on the decoder side we have the convolutional layers in the sizes [512,256,128,64,32]

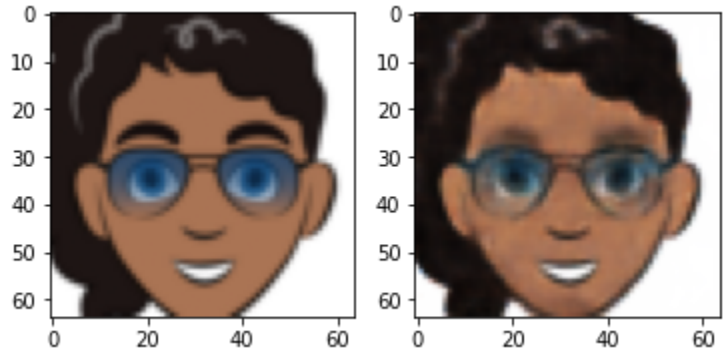


Figure 2 : The reconstructed images using the VAE
The above image shows the output of our VAE image reconstruction. The left image is the original image and the right image is the one reconstructed from the latent space.

Normalizing flows using Causal VAEs: An introduction

We'll walk you through the theory and basic steps to implement the "Deep Structural Causal Models for Tractable Counterfactual Inference" paper by Pawloski, Castro, et. al. Tractable inference on deep counterfactual models enables us to study causal reasoning on a per-instance rather than population level, which has valuable applications, for example, in estimating the extent of fairness of a model.

In "*Deep Structural Causal Models for Tractable Counterfactual Inference*," Pawlowski, Castro et. al. provide:

- "a unified framework for structural causal models using modular deep mechanisms"
- "an efficient approach to estimating counterfactuals by inferring exogenous noise via variational inference or normalising flows"

Here, we use a Causal VAE to implement the underlying machine learning algorithm harnessed by the power of normalizing flows. We model the pixels of an image using this Causal VAE and apply normalizing flows in intermediate steps to reconstruct a given image from the Google Cartoon Dataset. Future work would include extending this approach to other categorical variables in the dataset and performing counterfactual inference.

In the context of this tutorial, we train and test on the n=10k dataset.

Using Normalizing Flows with Structural Causal Models

Normalizing flows enable us to model complex probability distributions by transforming base distributions using the same dimensionality. As we'll see, this is accomplishable using tractable methods for modeling noise variables ε. Let's set up the problem:

For the invertible, explicit case: For an observed variable x, a diffeomorphic (gradient preserving) transformation f, and base variable ε ~ P(ε) such that x = f(ε), the output density p(x) can be computed as $p(x) = p(\varepsilon)|\det \nabla f(\varepsilon)|^{-1}$, evaluated at $\varepsilon = f^{-1}(x)$. We have here our standard change of variable formula applied to the markov kernels of Pearl structural causal models.
Equation 1:

$$x_i := f_i(\epsilon_i; \mathbf{pa}_i), \qquad p(x_i \,|\, \mathbf{pa}_i) = p(\epsilon_i) \cdot |\det \nabla_{\epsilon_i} f_i(\epsilon_i; \mathbf{pa}_i)|^{-1} \big|_{\epsilon_i = f_i^{-1}(x_i; \mathbf{pa}_i)} . \qquad (1)$$

For the amortised, explicit case: Since all intermediate transformations in the above formulation of the problem occur in the original (high dimensional) space, we find ourselves looking for more tractable methods. As suggested by Palowski, Castro, et. al., we can use arbitrary functional forms for the structural assignments in the causal model, which will loses us the ability to perform inversions and tractable likelihoods for modeling $(x_{
k} | \bold pa_{k]$. To tackle this, the authors suggest separating the assignments $\italic f_k$ into a 'low-level' invertible component $\italic h_{k}$,and a 'high level' non-invertible component $\italic g_{k}$, where ϵ_{k} is decomposed into two units, ε =(u_{k},z_{k}) such that:

Equation 2:

$$x_k \coloneqq f_k(\epsilon_k; \mathbf{pa}_k) = h_k(u_k; g_k(z_k; \mathbf{pa}_k), \mathbf{pa}_k), \qquad P(\epsilon_k) = P(u_k)P(z_k)\,. \tag{2}$$

One implementation choice for these methods is that of a convolutional neural network to model $\italic g_{k}$ and location and scale transformation performed by $\italic h_{k}$.
However, we come up against the same intractability problem as the conditional likelihood because z_{k} cannot be marginalized out to calculate P(x_{k} | \bold pa_[k}$.

We can then introduce a variational distribution $\italic Q(z_{k} | x_{k}, \bold pa_{k})$ which we can use as a lower bound on the true marginal conditional log-likelihood, that instead we will be maximizing. We will be using a variational autoencoder to realize this task.

Equation 3:

$$\log p(x_k \,|\, \mathbf{pa}_k) \geq \mathbb{E}_{Q(z_k|x_k,\mathbf{pa}_k)}[\log p(x_k \,|\, z_k, \mathbf{pa}_k)] - D_{\mathrm{KL}}[Q(z_k \,|\, x_k, \mathbf{pa}_k) \| P(z_k)]\,. \tag{3}$$

We calculate the argument of the expectation above analogously to Equation (1).

Equation 4:

$$p(x_k \,|\, z_k, \mathbf{pa}_k) = p(u_k) \cdot |\det \nabla_{u_k} h_k(u_k; g_k(z_k, \mathbf{pa}_k), \mathbf{pa}_k)|^{-1} \big|_{u_k = h_k^{-1}(x_k; g_k(z_k, \mathbf{pa}_k), \mathbf{pa}_k)}\,. \tag{4}$$

In our implementation, we use a variational autoencoder where the encoder function $e_{k}(d_{x}; \bold pa_{k}$ that outputs the parameters of a simple distribution over z_{k}.

Implementation Details

The basic idea of our implementation is to set up the variational autoencoder by creating an encoder and decoder that both have μ and logit scale heads as their final two layers. Configuring the heads of the encoder and decoder is accomplished as below:

```
class DeepIndepNormal(_DeepIndepNormal):
    def __init__(self, backbone: nn.Module, hidden_dim: int, out_dim: int):
        super().__init__(
            backbone=backbone,
            mean_head=nn.Linear(hidden_dim, out_dim),
            logstd_head=nn.Linear(hidden_dim, out_dim)
        )

class Conv2dIndepNormal(_DeepIndepNormal):
    def __init__(self, backbone: nn.Module, hidden_channels: int,
                 out_channels: int = 1):
        super().__init__(
            backbone=backbone,
            mean_head=nn.Conv2d(hidden_channels, out_channels=out_channels,
                                kernel_size=1),
            logstd_head=nn.Conv2d(hidden_channels, out_channels=out_channels,
                                  kernel_size=1)
        )
```

The variational autoencoder includes an encoder than takes in the variables $\bold x$ and predicts a μ and logit scale of the latent variational distribution $\italic Q(\bold z | \bold x)$. The DeepIndepNormal() function takes the “hidden” latent representation returned by the encoder and runs it through additional latent_layers that will be especially useful in the next iteration of the project, where we condition on a set of context variables and infer noise for evaluating a counterfactual. The output of these latent_layers are connected with two fully connected layers which act as the two heads of the resulting neural network, returning a μ and logit scale of the latent variational distribution $\italic Q(\bold z | \bold x)$.

```
self.encoder = Encoder(self.latent_dim, self.hidden_dim)
latent_layers = torch.nn.Sequential(torch.nn.Linear(self.latent_dim +
                    self.context_dim, self.latent_dim), torch.nn.ReLU())
self.latent_encoder = DeepIndepNormal(latent_layers, self.latent_dim, self.latent_dim)
self.decoder = Decoder(self.latent_dim + self.context_dim,self.hidden_dim)
self.decoder = Conv2dIndepNormal(self.decoder,3, 3)
```

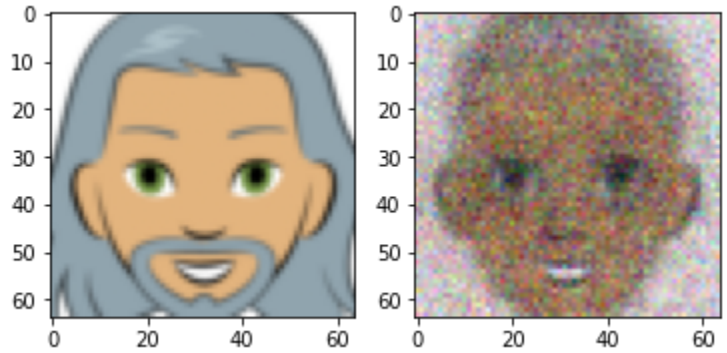
The convolutional neural network decoder receives the latent distribution $Q(\bold z | \bold x)$, represented by a $/mu_{z}$ and logit scale, and predicts the $P(\bold x | \bold z)$. Finally an AffineTransform is applied to the predicted distribution (of type TransformedDistribution), which can be represented as the following element-wise transformation:

$$x = \mu_{\hat x} + \sigma_{\hat x} \odot \bold x$$

```
x_pred_dist = self.decoder.predict(latent)
x_base_dist = Normal(self.x_base_loc, self.x_base_scale).to_event(3)

if isinstance(x_pred_dist, Independent):
    x_pred_dist = x_pred_dist.base_dist
    x_reparam_transform = AffineTransform(x_pred_dist.loc, x_pred_dist.scale, 3)

return TransformedDistribution(x_base_dist, ComposeTransform([x_reparam_transform]))
```



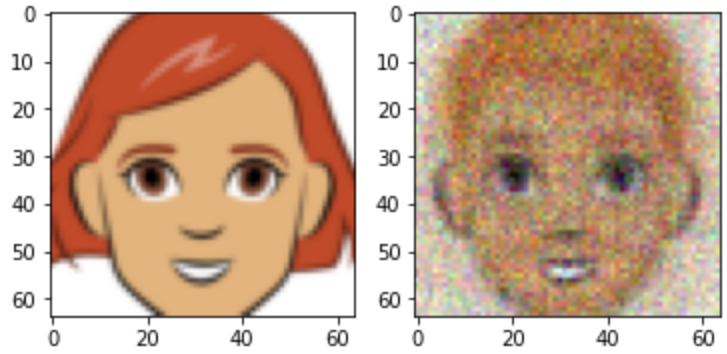


Figure 3 : Images reconstructed from the Causal VAE with Pyro Normalising Flows.

In Closing

We were able to implement a Causal VAE with the normalizing flows structure on a 10k set of faces from the Cartoon Dataset. We had lukewarm results in that the faces could be reconstructed only at a blurry level with some key features being altered. Because the Causal VAE on its own, without using normalizing flows, produced good results, we suspect that improvements could be made on the normalizing flows structure, such as perhaps in using batch normalization via affine transforms and using affine transforms to otherwise preprocess the data, as in the paper. Furthermore, it would be worth exploring other options such as the RealNVP model that the paper also does reference.

Future iterations on this project also include incorporating other endogenous variables into the SCM. Additionally, another important improvement would be implementing counterfactual inference to create images from existing images that represent counterfactual queries, such as “What would this face look like had they been wearing blue glasses?”

Using deep counterfactual inference in SCMs has a lot of interesting and valuable applications in transparency, fairness, and robustness in deep learning as well as in developing decision-support systems and personalized medicine. We hope you enjoyed this tutorial and keep on keepin’ on in your Causality in ML journey!